Energy Proportionality for Disk Storage Using Replication

Jinoh Kim and Doron Rotem
Lawrence Berkeley National Laboratory
University of California, Berkeley, CA 94720
{jinohkim,d_rotem}@lbl.gov

***Abstract*—Energy saving has become a crucial concern in datacenters as several reports predict that the anticipated energy costs over a three year period will exceed hardware acquisition. In particular, saving energy for storage is of major importance as storage devices (and cooling them off) may contribute over 25 percent of the total energy consumed in a datacenter. Recent work introduced the concept of energy proportionality and argued that it is a more relevant metric than just energy saving as it takes into account the tradeoff between energy consumption and performance. In this paper, we present a novel approach, called FREP (Fractional Replication for Energy Proportionality), for energy management in large datacenters. FREP includes a replication strategy and basic functions to enable flexible energy management. Specifically, our method provides performance guarantees by adaptively controlling the power states of a group of disks based on observed and predicted workloads. Our experiments, using a set of real and synthetic traces, show that FREP dramatically reduces energy requirements with a minimal response time penalty.**

I. Introduction

Power savings in datacenters has recently gained a lot of interest because of the costs involved in power delivery and system cooling. In a recent report to congress [1], EPA stated that many datacenters have already reached their power capacity limit and more than 10% of datacenters will be out of power capacity by the end of this year, while 68% expect to be at their limit within the next three years. In addition, another recent report [2] revealed that power occupies nearly a quarter of monthly operational costs in a datacenter, and if we consider power-related costs such as power distribution and cooling additionally, it makes up over 40% of the operational costs.

Among the many components in the datacenter, storage is the next largest consumer of energy after servers and cooling systems. It is currently estimated that disk storage systems consume about 25–35 percent of the total power [3]. This percentage of power consumption by disk storage systems will only continue to increase, as data intensive applications demand fast and reliable access to on-line data resources. This in turn requires the deployment of power hungry faster (high RPM) and larger capacity disks.

Several energy saving techniques for disk-based storage systems have been introduced in the literature [4], [5], [6], [7], [8]. Most of these techniques use the idea of spinning down the disks from their usual high energy consumption mode into a lower energy mode (sleep/standby mode) after they experience a period of inactivity whose length exceeds a certain threshold (*idleness threshold*). The reason for this is that typical disks consume about one tenth of the power in standby mode as compared with their power consumption when spinning.

There are several challenges associated with these spin-down techniques when applied to individual disks:

- *Energy and response time penalty*: Disks can only service requests while they are spinning, in case a request arrives when the disk is in sleep mode there is a response time penalty (typically 10–15 seconds) before the request can be serviced. In addition, considerable amount of energy is required to spin up the disk, in some cases this can exceed the energy saved by transitioning the disk to standby mode.
- *Expected length of inactivity periods*: Under many typical workloads found in scientific and other applications, individual disks do not experience long enough periods of inactivity (longer than the idleness threshold) thus limiting the opportunities to save energy.

Achieving *energy proportionality* in datacenters, rather than just energy saving, has been recently getting attention from industry and researchers and proposed as an important design metric [13]. The core principal behind energy proportionality is that computing equipment (storage, servers, networks, etc.) should consume power in proportion to their load level, i.e., a computing component that consumes x watts at full load, should consume $x \cdot \frac{p}{100}$ when running at p-% load.

The energy management approach we consider in this paper promotes energy proportionality and is different from existing approaches, as it is based on handling energy management in a *group* of disks rather than controlling disks individually. We show that our energy management is scalable to large datacenters with thousands of disks and preserves important features of the storage system such as parallelism and fault tolerance. As explained later, our approach exploits *data replication* which is used in many datacenters for reasons such as fault tolerance and load balancing. Popular distributed file systems, such as HDFS (Hadoop Distributed File System) [9] and GFS (Google File System) [10], also automatically replicate data by default. Data replication can help saving energy because when a data item is replicated several times, there is often an opportunity to select a replica found on a currently spinning disk, thus avoiding the costs (spin-up energy) involved in

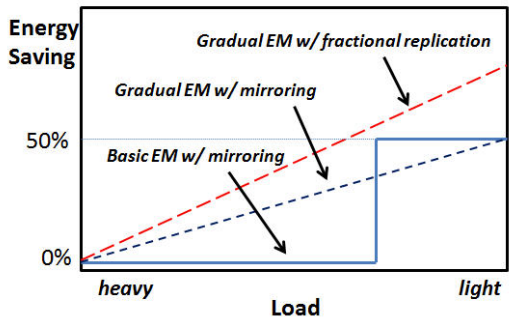


Fig. 1. Energy management (EM) models: Basic EM with mirroring works like "binary", either 100% disks running or 50% disks running. Gradual EM with mirroring can spin down disks individually, hence "gradual". However, the upper limit of energy saving is constrained to 50% even at extremely low load. EM with fractional replication (our approach) provides more flexible energy management, and the maximum energy saving can be selected by configuration.

accessing a replica found on a disk which is currently in sleep mode.

Although replication requires additional storage space, it is a relatively cheap resource as it is reported that storage resources in datacenters are often considerably under-utilized and use only a small fraction of the total available capacity (less than 25% according to several studies) [3], [11], [12]. In this paper, we present a novel replication strategy that achieves energy benefits while maintaining performance and fault tolerance. In particular, our *fractional replication* presented in Section III enables flexible gradual energy management based on workloads which promotes energy proportionality.

Figure 1 illustrates energy management (EM) models based on mirroring and our replication technique. With mirroring, disk contents are copied from one disk to another. Thus, mirroring can spin down up to half of total disks. As shown in Figure 1, typical mirroring can achieve up to 50% energy saving under certain conditions. eRAID [12] is an example of this. Disk mirroring with gradual EM model is more flexible by controlling disks individually. The work by Lang et al. [11] is based on this EM model. The EM model is more flexible as it controls disks individually, but it is still limited to a maximum of 50% energy saving. Naturally, replicating with a greater replication factor (e.g., 3) enables further energy saving (e.g., 67%), but such techniques are still rigid, lacking gradual adaptation to workloads. Our replication model is more flexible, and the maximum energy saving can be selected by configuration. We discuss this in more detail in Section III.

Another important challenge is to determine when to transition the disks to lower or higher power states. One typical approach for this is to use a set of thresholds for a load metric. For example, Lang et al. [11] use CPU utilization as their load metric, and assume that disk power states are changed at some threshold points of the load metric. Similarly, PARAID [5] monitors disk utilization and spin disks up/down based on some thresholds. In addition, workload characteristics can be significantly different among datacenters due to the type of applications they run, and also can change over time in the same datacenter.

In this paper, we present a systematical approach for energy management. Our solution, called FREP (Fractional Replication for Energy Proportionality), considers the disks in each disk array as a single unit which can be spun down or up. We refer to spinning up and down of such units as "gear-shift". As we will discuss, our gear-shift mechanism incorporates both predictive and reactive mechanisms, rather than relying on static thresholds. To improve performance, we maintain load balancing among the disk arrays that are currently spinning by access request re-direction. For *down-shifts* (for energy saving), we make *predictions* based on past historical information. For *up-shifts* (for performance guarantees), we utilize *reactive* information in order to respond to any performance degradation quickly. As a basis for these, FREP provides a replication strategy (and its associated functions). We believe that FREP is an important tool for achieving *energy proportionality* in the storage system.

Our main contributions in this paper are summarized as follows:

- We present basic functions and strategies for FREP energy management, including replication strategy, load distribution, and update consistency.
- We present a prediction model based on past historical observations with a de Bruijn graph to enable probabilistic decisions. In addition, we present our constraint-based gear-shift mechanism, by which FREP can shift gears for energy management.
- We provide extensive evaluation results with a diverse set of traces, including two Cello99 traces [14] 6-month apart each other, two OLTP traces [15], and synthetic traces with different workload characteristics.
- We also show that our gear-shift mechanism can be used to enhance PARAID-type systems in terms of energy management with performance guarantees.

This paper is organized as follows. In Section II we present some related studies that use replication for energy savings. In Section III, we introduce the FREP replication strategy, and a series of functions for I/O service and energy management in such a replicated environment. Our prediction model with de Bruijn graphs is introduced in Section IV, where FREP gear-shift mechanism based on the prediction model is also described. Our extensive experimental results with diverse workload sets are presented in Section V. Finally, summary and conclusions are presented in Section VI.

II. Related Work

Our work is inspired by Power-aware RAID (PARAID) [5] which was the first work to introduce the concept of gear-shifting based on system load. It provides a replication strategy, called skewed striping, for disk energy management without service disruption. The main difference is that, PARAID shifts gears within a RAID unit by spinning up/down one or more disks in the array, while we do it across multiple RAID arrays. Another difference is the conditions leading to

a gear-shift. PARAID takes a reactive approach based on disk utilization with thresholds. Our gear-shift mechanism uses a prediction technique based on past history in addition to a reactive technique.

Lang et al. [11] used mirroring for disk energy management. Traditionally, mirroring gives two options — running all the disks (100%) or half of disks (50%). The authors present gradual disk power control combined with load balancing techniques by using a new replication strategy, called chained declustering. Although this new technique provides more flexibility, energy saving is still limited to 50%, since at most 50% of disks can be spun down. However in real systems the degree of load variation can be more dramatic. For example, in [16], the authors observed a high degree of load variation, over a factor of three in a commercial web site. Since datacenters usually tend to over provision resources to satisfy peak loads, there may be many opportunities to save energy by factors much greater than 50%.

Although energy management is a crucial problem for datacenters, performance guarantees may be even more important. Hence, energy management needs to be performed within acceptable performance bounds. As pointed out in [17], simple dynamic energy management techniques, such as timeout-based disk spin-down, need to pay significant performance penalty. This makes administrators of datacenters reluctant to employ such approaches in these cases where system performance is a crucial requirement. To provide energy saving within a controlled performance environment, several research works have taken system SLAs into account. Hibernator [7] employs response time constraint, and considers an optimization problem to minimize energy subject to a given constraint. Similarly, eRAID [12] uses a response time constraint in addition to a system throughput constraint for their energy saving problem. However, we observed that average response times can experience a very high degree of variance, sometimes exceeding three orders of magnitude. Elnozahy et al. [18] employ a "percentile-based response time" to specify the performance constraint for Web servers. In this work, we also employ this to define system SLAs.

Both static and dynamic techniques have been studied for workload-adaptive energy management. A well known static technique, which we call *2-competitive* algorithm [4], is based on transitioning the disk to sleep mode whenever it experiences a period of inactivity greater than $\frac{\beta}{P_\tau}$ where β is the energy penalty (in Joules) for having to serve a request while the disk is in sleep mode (i.e., spinning the disk down and then spinning it up in order to serve a request) and P_τ is the rate of energy consumption of the disk (in Watts) when spinning. This technique does not attempt to predict the workload and may sometimes lead to unstable performance. Dynamic techniques include employing a multiple set of "experts" [19], [20]. In [19], a set of timeout values are combined to determine the next idleness timeout based on associated weights varied over time, based on the past history. In [20], rather than using an aggregated result, one expert is chosen for energy management, whenever needed, based on the weight values. In updating weight values, this work considered both energy saving and response time latencies. Chung et al. [21] established Markov chains for dynamic energy management, and calculated state transition probabilities based on observations for nonstationary workloads. Energy management actions are determined based on the probabilities. Our prediction model also refers to past observations for workload adaptability.

III. System Model

We introduce our system model and our energy management solution called FREP (Fractional Replication for Energy Proportionality). We are particularly interested in *read-many, write-rare* environments, as many datacenters use write off-loading [22] or Log Structured Files techniques [23] to batch together write transactions and minimize their effect on power consumption.

FREP manages power states on the basis of a group of disks (e.g., a RAID array). In other words, a group of disks (which form a RAID array) are transitioned together to either *standby* or *active* state in the course of energy management. We assume that the entire disks in a group are either in a standby state (non-spinning), or they are all spinning in the active state.[1]

Formally, we define *node* as an array of disks managed together with respect to energy management. Thus, a node is a collection of disks and there is no disk sharing between nodes. For example, a node can be a RAID-5 array that includes data and parity disks. For scalability, a large storage system can be divided into multiple disjoint *partitions*, each of which consists of its own set of participating nodes. In the rest of this paper, all functions for energy management and analysis refer to a single partition. A partition of a storage system consists of a set of nodes $N = \{N_i\}$. We distinguish between two classes of nodes: Covering Set (*CS*) nodes that are always spinning and contain between them a copy of each data item in the partition, and *non-CS* nodes that can change their power states for energy management purposes. For ease of exposition, we assume n nodes in total, where the first m ($1 \leq m < n$) nodes with the lowest indexes are CS nodes, i.e., $\{N_1, N_2, .., N_m\}$, and the rest are non-CS nodes, i.e., $\{N_{m+1}, N_{m+2}, .., N_n\}$. Table I summarizes notations used in this paper.

Figure 2 illustrates our gear-shift model, from the lowest gear level to the highest gear level. In the figure, filled nodes are active, whereas non-filled nodes are standby. In the lowest gear level, only CS nodes are active (disks associated with CS nodes are spinning) while at the highest gear level all the nodes are active. As explained later, for any node in standby mode, all requests to its data are redirected and serviced (in a balanced fashion) by other active nodes that hold an associated replica. Our replication enables continuous service regardless of energy management with minimal storage requirement, as discussed in the next section.

Figure 3 shows an example of a storage system with multiple partitions with different configurations. As shown

[1]We interchangeably use "standby/active", "spun-down/spun-up", and "powered-off/powered-on" for disk array state.

TABLE I
NOTATION

Symbol	Description
N_i	A node with index i ($N_i \in N$)
D_i	Original data for N_i
$D_i(\frac{a}{b})$	a/b fraction of D_i
V_i	Storage volume of N_i
$\mathcal{W}_i$	Replica storage for N_i
n	Total number of nodes ($= \|N\|$)
m	Number of CS nodes
$n-m$	Number of non-CS nodes
w	Number of active nodes
$n-w$	Number of standby nodes
C	Storage capacity
ρ	Storage utilization
$L_i(w)$	Load for node i with w active nodes
LIF	Load imbalance factor; $LIF = 0$ means balanced load
$R(p)$	p-% response time in ascending order in a time frame

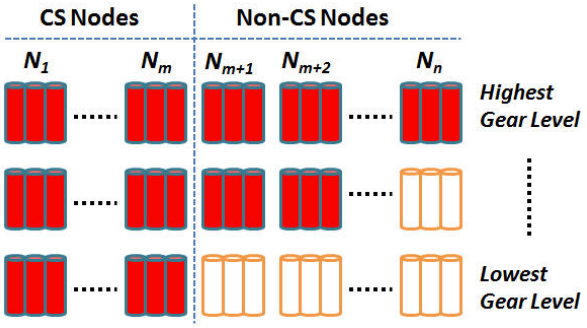


Fig. 2. FREP gear shift model: Gear-shift takes place based on the given performance constraint. In the highest gear level, entire nodes are active, whereas only a small number of nodes (CS nodes) are active in the lowest gear level. The number of CS nodes is configurable. Filled nodes are active and empty nodes are standby in this figure.

in the figure, each partition can be configured with different number of CS and non-CS nodes. As mentioned, our energy management functions are effective within a partition.

A. Replication strategy

We present our replication strategy that enhances energy proportionality. The main idea behind FREP is to utilize data replication in order to avoid performance penalties. We will show that our replication scheme can achieve continuous data availability even in energy saving mode. On the other hand, energy management without replication usually causes severe latencies because of the need to spin up disks from standby node, an operation that can take tens of seconds to get back to service (hence unacceptable to datacenters in general). For example, as shown in the evaluation section, a simple energy management technique based on the *2-competitive algorithm* [4] mentioned earlier, may sometimes incur response time penalty causing performance degradation by a factor of 10.

Next we outline the general structure of our replication

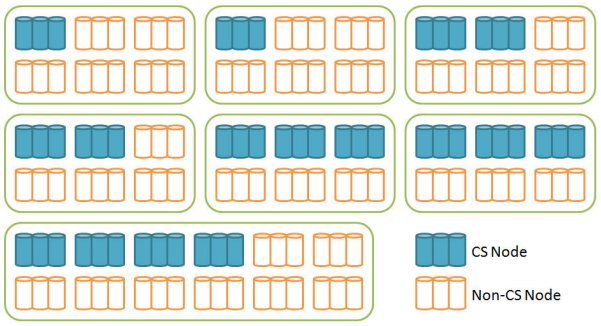


Fig. 3. FREP partitions

	CS nodes		non-CS nodes			
Node	N_1	N_2	N_3	N_4	N_5	N_6
Orig.	D_1	D_2	D_3	D_4	D_5	D_6
Repl.	$D_3(\frac{1}{2})$	$D_3(\frac{1}{2})$				
	$D_4(\frac{1}{2})$	$D_4(\frac{1}{2})$	$D_4(\frac{1}{3})$			
	$D_5(\frac{1}{2})$	$D_5(\frac{1}{2})$	$D_5(\frac{1}{4})$	$D_5(\frac{1}{4})$		
	$D_6(\frac{1}{2})$	$D_6(\frac{1}{2})$	$D_6(\frac{1}{5})$	$D_6(\frac{1}{5})$	$D_6(\frac{1}{5})$	
			$D_1(\frac{1}{4})$	$D_1(\frac{1}{4})$	$D_1(\frac{1}{4})$	$D_1(\frac{1}{4})$
			$D_2(\frac{1}{4})$	$D_2(\frac{1}{4})$	$D_2(\frac{1}{4})$	$D_2(\frac{1}{4})$

Fig. 4. FREP replication scheme with six nodes

scheme. We assume that before replication is introduced each node N_i has some original data denoted by D_i. We denote by, $D_i(\frac{a}{b})$ an a/b fraction of D_i. After replication each node will hold some replicated data in addition to its original data as follows: (a) Each CS node gets an equal fraction of the replicated data from each non-CS node; (b) For fault tolerance and load balancing purposes, non-CS nodes maintain replicas of original data associated with CS nodes. Again, each non-CS node gets an equal fraction of the replicated data from each CS node; (c) Non-CS nodes keep replicas of original data (D_i's) from specific other non-CS nodes in a skewed way as explained later. We call cases (a) and (b) *balanced-replication* and (c) *skewed-replication*.

Figure 4 illustrates an example of our replication scheme with 6 nodes (in a partition), two of which are CS nodes. Since there are two CS nodes in this setting, they each keep a disjoint half of non-CS node data. Non-CS nodes also maintain disjoint replicas of CS node data. As there are four such nodes each gets a disjoint one quarter of the data. This replication is done strictly for fault tolerance and performance as will be discussed in Section III-F. In addition, non-CS nodes maintain a part of other non-CS node replicas based on the gear-shift principle. The replication between non-CS nodes helps to distribute the request load in energy saving mode (i.e., a non-highest gear level).

We next explain the skewed-replication, the replication scheme used between non-CS nodes. The original data D_i of a non-CS node N_i ($i > m+1$) are replicated equally to other non-CS nodes with lower indexes, i.e., N_j for $m+1 \leq j < i$, in a random and disjoint fashion. This is done as follows. For each j for ($m+1 \leq j < i$), we randomly select $\frac{1}{i-(m+1)}$ of the blocks of D_i (original data of non-CS node N_i) without

replacement, and copy them to node N_j.

B. Storage requirements

Now, let us discuss the storage requirement for replicas for each node ($\mathcal{W}_i$). Let V_i be the size in bytes of D_i, i.e., the data volume of N_i. The following equation shows the storage requirement for replicated data on each node.

$$\mathcal{W}_i = \begin{cases} \sum_{k=m+1}^{n} V_k/m & \text{if } 1 \le i \le m; \\ \sum_{k=1}^{m} V_k/(n-m) & \text{if } i = n; \\ \mathcal{W}_{i+1} + V_{i+1}/i & \text{otherwise.} \end{cases} \quad (1)$$

In the equation, the first case is for CS nodes, and the second case is for the last non-CS node N_n. These nodes only hold replication data resulting from balanced-replications. The third case is a recursive expression for the storage requirements resulting from the skewed-replication for non-CS nodes described above (except for non-CS node N_n).

We next discuss the total storage requirement for FREP. For simplicity, from now on, we assume that each node holds the same volume of original data, i.e., $\forall N_i \ V_i = V$, so that the total storage for original data is nV. Clearly storage requirements are at least $2nV$ as each item is replicated at least once, the next proposition shows that it is less than $3nV$.

Proposition 3.1: The total storage requirement $\mathcal{W}$ for FREP is $\mathcal{W} \approx 3nV - mV(1 + \ln \frac{n}{m})$.

Proof: Since CS nodes maintain a whole replica for non-CS nodes and non-CS nodes also maintain a whole copy for CS nodes, $\mathcal{W} = 2nV + \alpha$, where α is the storage requirement for skewed-replication (i.e., replication between non-CS nodes). For each non-CS node N_i, the storage requirement for its replica is $\frac{i-(m+1)}{i-1} \cdot V$, for $i \ge m+1$, and hence, the additional storage α is,

$$\begin{aligned} \alpha &= \sum_{k=m+1}^{n} \frac{k-(m+1)}{k-1} \cdot V = V \sum_{k=m}^{n-1} \frac{k-m}{k} \\ &= V \left[\sum_{k=m}^{n-1} 1 - m \sum_{k=m}^{n-1} \frac{1}{k} \right] \end{aligned} \quad (2)$$

For a harmonic number $H_{n-1} = \sum_{k=1}^{n-1} \frac{1}{k}$, it is approximate to $H_{n-1} \approx \ln n + \gamma$, where γ is Euler's constant. Hence, $\sum_{k=m}^{n-1} \frac{1}{k} = \sum_{k=1}^{n-1} \frac{1}{k} - \sum_{k=1}^{m-1} \frac{1}{k} = H_{n-1} - H_{m-1} \approx \ln n + \gamma - (\ln m + \gamma) = \ln \frac{n}{m}$.

$$\begin{aligned} \alpha &\approx V \left[n - m - m \ln \frac{n}{m} \right] \\ &= V \left[n - m(1 + \ln \frac{n}{m}) \right] \end{aligned} \quad (3)$$

Therefore, the maximum storage requirement $\mathcal{W}$ is:

$$\mathcal{W} = 2nV + \alpha \approx 3nV - mV(1 + \ln \frac{n}{m}) \quad (4)$$

■

Now, let us consider possible CS/non-CS node configurations from the storage perspective. For simplicity, we assume that all disks in a partition have an equal disk capacity C.

Proposition 3.2: The number of CS nodes m is bounded by: $\rho n \le m \le m^*$, where ρ is storage utilization and m^* is the largest m that satisfies the inequality: $1 + \frac{m}{n-m} + \ln\left(\frac{n}{m+1}\right) \le \frac{1}{\rho}$.

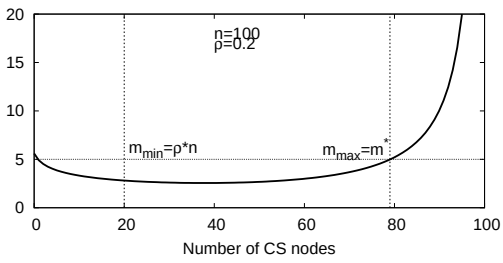


Fig. 5. Constraints of number of CS nodes: Based on the constraints $\rho n \le m \le m^*$ (Proposition 3.2), it is possible to compute the min/max number of CS nodes for any configuration. In this configuration with $n = 100$ and $\rho = 0.2$, the min/max number of CS nodes are 20 and 79, respectively.

Proof: We first derive the minimum number of CS nodes (m_{min}). Any CS node needs to keep (i) its own copy, and (ii) replicas for non-CS node. By assumption, (i) = ρC, and (ii) = $\rho C \cdot \frac{n-m}{m}$. This cannot not exceed the capacity C. Therefore,

$$\begin{aligned} \rho C + \frac{\rho C(n-m)}{m} &\le C \\ \rho n &\le m \end{aligned} \quad (5)$$

To obtain the maximum number of CS nodes (m_{max}), we can conversely consider the minimum number of non-CS nodes (o_{min}) that are required for replication. Then, $m_{max} = n - o_{min}$. Among non-CS nodes, N_{m+1} requires maintaining the largest space for replication, and hence, o_{min} depends on the space availability of N_{m+1}. N_{m+1} maintains (1) its own copy, (ii) CS node replicas, and (iii) non-CS nodes replicas. We can simply compute that (i) = ρC and (ii) = $\rho C \cdot \frac{m}{n-m}$ and (iii) = $\rho C(\frac{1}{m+1} + ... + \frac{1}{n-1}) = \rho C \sum_{k=m+1}^{n-1} \frac{1}{k} = H_{n-1} - H_m \approx \rho C \ln \frac{n}{m+1}$.

Since the summation of (1)+(ii)+(iii) should be less than C, we obtain:

$$\begin{aligned} \rho C + \rho C \cdot \frac{m}{n-m} + \rho C \cdot \ln\left(\frac{n}{m+1}\right) &\le C \\ 1 + \frac{m}{n-m} + \ln\left(\frac{n}{m+1}\right) &\le \frac{1}{\rho} \end{aligned} \quad (6)$$

We define m^* as the maximum m that satisfies Equation 6. Combining it with Equation 5, finally,

$$\rho n \le m \le m^* \quad (7)$$

■

Figure 5 shows an example for possible min/max number of CS nodes for a system with $n = 100$ and $\rho = 0.2$. In the figure, the minimum required number of CS nodes is 20 and the maximum is 79. In other words, the number of CS nodes cannot be smaller than 20 or greater than 79 to successfully accommodate the FREP replication. From the perspective of energy management, the maximum energy saving can be obtained up to 80% ($1 - \frac{20}{100}$) with the minimal CS node setting in this example.

	CS nodes		non-CS nodes			
w	N_1	N_2	N_3	N_4	N_5	N_6
6	1	1	1	1	1	1
5	1.2	1.2	1.2	1.2	1.2	—
4	1.55 (1.5)	1.55 (1.5)	1.45 (1.5)	1.45 (1.5)	—	—
3	2.11 (2)	2.11 (2)	1.78 (2)	—	—	—
2	3	3	—	—	—	—

Fig. 6. Example of load distribution: This example shows load before and after (in parenthesis) optimization, as a function of the number of active nodes (w). With a lower gear, CS nodes have greater load than non-CS active nodes. This can be mitigated by our optimization with probabilistic redirection.

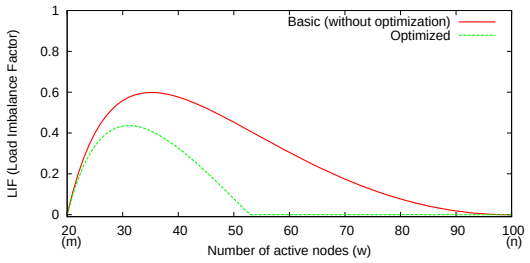


Fig. 7. Load imbalance factor (LIF) for CS node ($LIF = given\_load - balanced\_load$): With the optimization, LIF is significantly reduced.

C. Load distribution

We next discuss the impact of energy management on load balancing and how we distribute the load to active nodes by taking advantage of the replication. In FREP, active nodes take up load from standby nodes, based on the location of replicas (the details on implementation of request redirection for standby nodes are described in Section III-E). For example, in Figure 4, nodes N_1–N_5 service requests for N_6 when N_6 is in standby, since they have N_6 replicas. Similarly, when N_5 transitions to standby, N_1–N_4 take up N_5 load evenly. Figure 6 shows an example of load distribution as a function of the number of active nodes (w). For simplicity, we assume that the load generated by accessing the original data in each node is uniform and normalize it to 1 (i.e., load=1). Note that numbers in parentheses in the figure represent the adjusted load achieved by our optimization algorithm to mitigate load imbalance between CS and non-CS nodes. We will discuss it later in this section.

Next we show how to compute the load for each node based on the number of active nodes. Let $L_i(w)$ be the load for node i where the number of active nodes is w ($m \leq w \leq n$). By definition, $L_i(w = n) = 1$ and $\sum_i L_i(w) = n$. We can then compute load as follows:

$$L_i(w) = \begin{cases} 0 & \text{if } i > w; \\ L_{i+1}(w) + 1/w & \text{if } m < i \leq w; \\ 1 + \frac{\sum_{k=w+1}^{n}(1 - \frac{w-m}{k-1})}{m} & \text{otherwise.} \end{cases} \quad (8)$$

In Equation 8, the first case is for standby nodes, and thus the load is necessarily zero. The second case is for active non-CS nodes, and can be defined recursively with the newly introduced load at every node spin-down ($1/w$). Alternatively, it can be defined (non-recursively) as $L_i(w) = \sum_{k=w+1}^{n} \frac{1}{k-1}$. The last case is for CS nodes. Since each active non-CS node takes $\sum_{k=w+1}^{n} \frac{1}{k-1}$ load from standby nodes, the rest of the load on standby nodes which is equal to $\sum_{k=w+1}^{n}(1 - \frac{w-m}{k-1})$ should be handled by CS nodes. Therefore, each CS node takes $\frac{\sum_{k=w+1}^{n}(1-\frac{w-m}{k-1})}{m}$ in addition to its own load (i.e., 1).

In Figure 6, we see some degree of load imbalance between CS and non-CS nodes, as non-CS nodes transition standby. This is the inherent characteristic of our load distribution algorithm. More accurately, we define a metric *load imbalance factor (LIF)* to express how the load deviates from the ideal balanced state: $LIF = given\_load - balanced\_load$. Thus, $LIF > 0$ means over-loaded, while $LIF < 0$ means under-loaded and $LIF = 0$ indicates perfect load balance. $LIF_i(w)$ represents LIF for node i where the number of active nodes is w. Hence, LIF can be expressed as $LIF_i(w) = L_i(w) - n/w$, since n/w is the ideal balanced load. Figure 7 shows LIF for CS nodes varying with the number of active nodes in a system with $n = 100$ and $m = 20$. We can see that load is balanced for the two extremes, i.e., $w = m$ or $w = n$. In between these extremes, we see some degree of load imbalance in the figure.

We now present an optimization technique to reduce LIF for CS nodes, thus bringing the system closer to a balanced state. The basic idea is to have non-CS nodes service CS node load based on replicated data they maintain. To achieve this, it is possible to redirect requests accessing CS node data to any active non-CS nodes probabilistically if the corresponding replicas are kept in such active non-CS nodes. In our optimization, we compute the redirection probability (θ) as follows:

$$\theta = \min\left(1, \frac{\sum_{k=1}^{m} LIF_k(w)}{m} \times \frac{n-m}{w-m}\right) \quad (9)$$

In the equation, $\frac{\sum_{k=1}^{m} LIF_k(w)}{m}$ is simply $LIF_1(w)$, since each CS node has the same LIF.

The intuition behind this is to redirect requests for any CS node data more aggressively if either the LIF for CS node is greater or the number of active non-CS nodes are smaller (or both). For any request accessing CS node data, we can probabilistically redirect the request based on the computed θ but only if any of active non-CS nodes keeps the replica. For example, suppose $\theta = 0.5$ and active non-CS nodes keep $1/2$ of CS node replicas. In that case, 50% requests to CS node data can be redirected to non-CS nodes (probabilistically), but 50% of them can actually be serviced by active non-CS nodes. Consequently, it can reduce CS node load by 0.25. Figure 7 shows LIF for CS nodes with and without the optimization in a system with $n = 100$ and $m = 20$. We can see that our optimization can significantly mitigate load imbalance compared to the basic one. Except for states with fairly small number of active non-CS nodes (less than 50 in the graph), load is almost balanced for active nodes.

D. Update consistency

Although our focus is more on *read-dominant* environments, FREP provides write functionality for new writes and update

consistency. The main principle for writes is that we redirect all write requests to CS node first, and perform synchronization in the *reorganization phase* later. Hence, the basic idea is similar to write off-loading [22], in which all writes to powered-off disks are temporarily redirected to a cache area for future synchronization.

In FREP, however, there is no assumption about extra cache. Instead, FREP redirects all writes to CS nodes *regardless of* power state of non-CS nodes. Recall that, by our replication strategy, any non-CS node data block is also maintained in CS node. For any update request, the CS node data/replica is updated, and the original block is simply marked as *stale* if it is in any non-CS node to prevent subsequent accesses. For new writes, one of CS nodes is selected to accommodate it. Note that these write redirections are only applied while the system is functioning in energy management mode. Otherwise, the original data is directly updated, and newly written blocks are disseminated (based on system policy).

Whenever the gear goes up to the highest level, a background *reorganization* process is scheduled. That is, any subsequent down-shift condition enables reorganization to be executed. During the reorganization, down-shift is postponed. In this phase, all stale blocks in non-CS nodes are synchronized with the corresponding copies of CS node. Newly written blocks in CS nodes can also be migrated to non-CS nodes based on system policy, and replication takes place with using *skewed-replication* scheme, as discussed in Section III-A. In contrast, non-migrated blocks (by system policy) are replicated to non-CS nodes (*balanced-replication*). The gear is then shifted down after completing reorganization if the down-shift condition is still effective. In the course of reorganization, any condition to shift up gears can interrupt this process, and the system devotes to user requests.

E. Request redirection

In this section, we describe data structure for meta information and the detailed algorithm for request redirection for both reads and writes. Since we discussed request redirection for load distribution and write procedures in the above two sections, we focus here on explaining general operations.

To enable request redirection, FREP maintains a mapping table, as shown in Figure 8. When a request arrives and the original data block is located in a standby node, FREP first refers to the mapping table. For each data block, the associated mapping table entry contains replica addresses in a non-CS node (NRA) and a CS node (CRA), in addition to a *flag* indicating update history (stale or new creation). The associated non-CS node address can be *null*, but CS node address should not be *null*. If NRA is not null and the node is active, the request is redirected to the NRA. Otherwise, the request is redirected based on CRA. Note that the mapping table only is for skewed-replication. For balanced-replication, there is no block-level meta information; instead, each node maintains <node#, start_block_offset, number_of_blocks>, and replica block address is calculated based on the information. This reduces storage requirement for meta information.

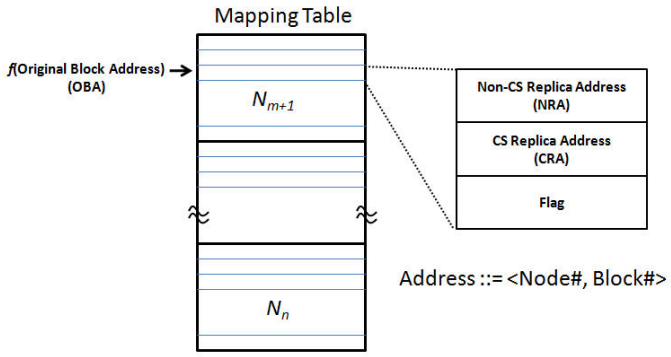


Fig. 8. FREP mapping table: The mapping table contains two replica addresses, one for non-CS node and the other for CS node, and a flag to tell staleness or new write for each data block for non-CS nodes. It is then used for redirection of requests.

```
Input: Request r
switch r.type do
    case new writes:
        Get free blocks from one of CS nodes;
        Write the block to the address;
        Create a mapping entry with flag = CREATE;
    endsw
    case update:
        Entry e = MappingTable.get(r.address);
        Write block to e.CRA;
        e.flag = STALE;
    endsw
    case read:
        Entry e = MappingTable.get(r.address);
        if e.flag == STALE then
            Read block from e.CRA;
        end
        else if OBA.node is active then
            Read block from e.OBA;
        end
        else if NRA is not null then
            Read block from e.NRA;
        end
        else
            Read block from e.CRA;
        end
    endsw
endsw
```

Algorithm 1: Request service

Algorithm 1 illustrates how the request is serviced in the system. If the request is for a *new write*, FREP allocates a block from one of CS node, and a new entry is created in the mapping table. The request for update is redirected to the associated CRA, and then the flag is set to stale. In the case of *read*, the stale bit is checked first. If the block is stale, the request is handled by the CRA. Otherwise, the request is handled by the priority OBA $>$ NRA $>$ CRA, as shown in the algorithm.

We consider the stripe size for the block size in the mapping table. If the system is configured with 146 GB disks, 128 KB for stripe size, and 24 nodes (this is our experimental configuration with 120 disks in Section V), the required

amount of storage is less than 230 MB for the mapping table.[2] Since server clusters is typically configured with large memory (as large as tens of GBs), the mapping table can be accommodated in the main memory.

F. Fault tolerance

Our main consideration in this paper is energy management, and changing disk power states we discussed are based on the system operating normally in non-failure mode. However, failure can happen for any components, and we need to handle them. In this section, we only consider the *fail-stop* model assuming failure events are detected immediately as they occur.

Our principle for fault tolerance is to immediately stop energy management functions. This means that standby nodes, if any, are spun up immediately, and all nodes other than the failed node start servicing access requests. If the failed node is a non-CS node, all requests to that node are redirected to CS nodes. In the case of CS node failure, non-CS nodes take over all requests to the CS node, since non-CS nodes maintain CS node replicas (as seen in Figure 4). In that case, load for non-CS nodes is $1 + \frac{1}{(n-m)}$, while active CS node have 1. In Algorithm 1, we omitted the procedure for request distribution under a failure environment for simplicity, but it can be easily considered.

This strategy allows us to configure a system even with a single CS node, if needed. However, in this case, there is no designated node to accommodate new write blocks. In such configuration, thus, we can simply have the non-CS node with the greatest identifier (i.e., N_n) accommodate them, since it has no duty of keeping any replicas (thus, there is a high possibility of available space). After recovering from failure, synchronization is performed as in the reorganization phase. In some cases, one additional task in this phase is the synchronization from non-CS nodes to the recovered CS node. In the next section, we discuss how FREP determines gear-shift conditions.

IV. Gear-shift Mechanism

FREP shifts gears for energy management, and as a result, its energy benefits and response time performance critically rely on the gear-switch mechanism. In this section, we present our gear-switch mechanism.

A. Service constraint

Our main goal in designing gear-switching mechanism is to maximize energy benefits, such that the system SLA is met. There are various SLA metrics in the literature. Average response time is one typical example [7], [12]. However, in real life trace logs we observed a high degree of variation for this metric (Cello99). As shown in Figure 9, variation of over 3 orders of magnitude is possible. The figure plots average response time observed with different window sizes from 1 minute to 1 hour. Even with a large time window, we can still see drastic changes across time. Such a high degree of variation makes it difficult to use this metric in defining system SLAs For this reason, we alternatively consider *percentile* for system SLAs.

[2]We compute this with 8-byte mapping entry size: NRA(31 bits), CRA (31 bits), flag (2 bit), and node# (7 bits) and Block# (24 bits) in the address format.

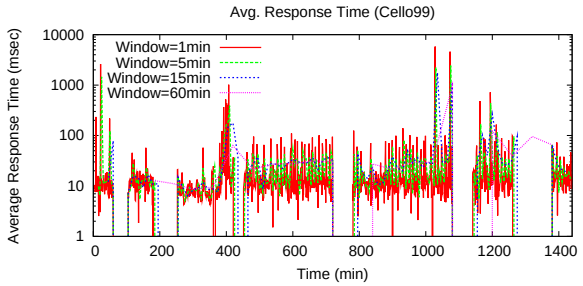


Fig. 9. Variation of Average Response Time (Cello99): Average response time is highly varied over 3 orders of magnitude. Note that the y-axis is in log scale.

Percentile is widely employed in definitions of system SLAs. For example, *availability* requirement (for data, node, etc) is often defined with x-nines, where x-nines refers to the number of '9' in percentile value. Thus, 5-nines refers to 99.999% availability. In this paper, we use percentile of request response time for the system SLA. For instance, we can specify a service constraint: "*99% of requests should be serviced within 500 msec.*" This is a safer metric than average response time, particularly for such environments with a high degree of variations, in which only smaller number of delayed completions can critically affect the aggregated result. Formally, a system SLA is defined:

$$SLA : R(p) \leq \tau \tag{10}$$

Here, p is a percentile, $R(p)$ is p-% response time in ascending order (observed in a given time interval), and τ is the response time constraint. Thus, we need to provide values for p and τ to specify an SLA. With the specified SLA, FREP checks whether p-% requests lie within τ. For this, FREP uses a predictive approach, and makes gear-shift decisions based on prediction. Before discussing our prediction model, we first discuss workload diversity with real and synthetic traces, and then continue to discuss our prediction model based on de Bruijn graphs.

B. Sensitivity to Workload Characteristics

Workload characteristics can be widely different for systems or even in a single system over time. Figure 10 compares request arrival rates from two traces, a Cello99 trace and an OLTP trace, each of which is from HP Storage Research Lab [14] and University of Massachusetts [15], respectively. We call these traces "Cello99" and "umass". The details of workload traces used in this paper are described in Section V. As can be seen in the figure, the traces have fairly distinctive patterns. The Cello99 trace looks highly bursty going over to 1,000 requests in a second, while the umass trace is relatively uniform moving up and down between arrival rate

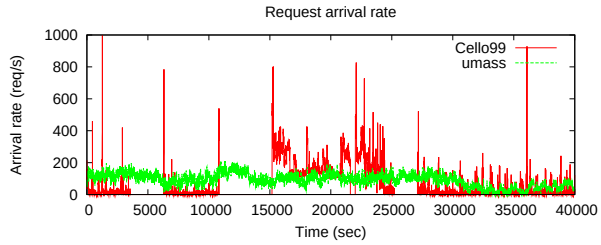


Fig. 10. Request arrival rates for real traces (Cello99 and umass): Workloads can have highly different characteristics. The Cello99 trace shows a fairly bursty characteristic, while the umass trace shows relatively uniform with respect to request arrival rate.

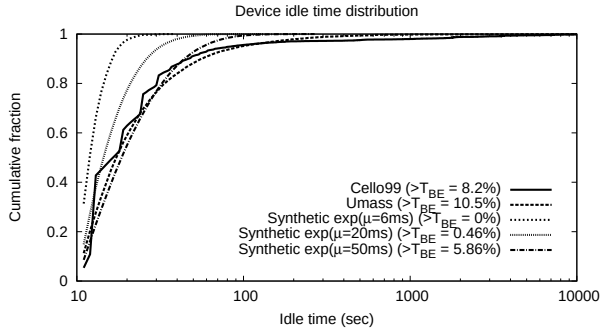


Fig. 11. Disk idle time distribution (Cello99, umass, synthetic): Disk idle time distributions are different for each trace, suggesting different opportunities for energy saving.

0 to 200 per second over time. Another recent observation also reported significant I/O workload differences for server systems, including mail, web, and file servers [24].

Figure 11 shows disk idle time distributions for the two traces described above and three new synthetic traces, each of which has an exponential distribution with μ=6ms, μ=20ms, and μ=50ms, respectively, for inter-arrival time. The first two synthetic traces were characterized in Hibernator [7] for OLTP workload (μ=6ms) and Cello99 workload (μ=20ms). We additionally create the third synthetic trace to represent a relatively light workload. Note that inter-arrival time for our cello trace is 29 ms, and for the umass trace is 8 ms. We assume that the disk is *idle* if it does not perform any action over 10 seconds. In addition, T_{BE} refers to *break-even time* for our disk model used in Section V.

In the figure, the Cello99 trace shows a heavy tail, indicating some devices experienced very long idle times. The umass trace looks similar to Cello99, but shows a slightly shorter tail. The synthetic traces show relatively short lengths of idle times and non-heavy tails compared to the real-world traces, and provide lesser opportunities for energy savings for the 2-competitive algorithm (which is based on a fixed idleness threshold). One interesting observation is that the synthetic trace with μ=50ms shows smaller opportunities than the real traces even with heavier arrival rates. These observations suggest that in order to get better results, we need energy savings strategies that consider workload characteristics.

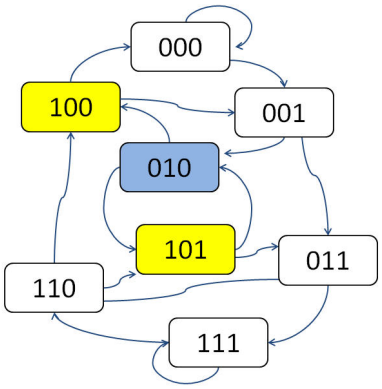


Fig. 12. A de Bruijn graph: With 3 bits, there can be 8 states in the graph.

Variations in workload characteristics indicates that static techniques should be ruled out. For example, if we simply apply the 2-competitive algorithm for the synthetic traces (particulary for the first two synthetic traces), there will be severe performance and energy penalties. Although the static parameters can be tuned for each workload, it is usually difficult to capture workload characteristics *a priori*. A *predictive* approach can be an option for dynamic adaptation to different workload characteristics. In this paper, we consider a predictive approach based on *probability* that is constructed based on past observations, as discussed next.

C. Prediction with de Bruijn graphs

We use "state-based" predictors to probabilistically predict future states by using a de Bruijn graph. In a de Bruijn graph with k bits, there exist 2^k states represented in binary, each of which has 2 incoming edges and 2 outgoing edges. For each state, one of two events can take place , *0 or 1*, based on how the current state transitions to the next state along with the corresponding outgoing edge. With this property, each state tells us what has happened over k time frames. For example, if the current state is '100', there was *1*-event before 2 time frames, followed by 2 consecutive *0*-events. Thus, we can be aware that the most recent event was *0*-event. Figure 12 illustrates a 3-bit de Bruijn graph with 8 possible states. In the figure, the current state is '010', and the next possible state is either '100' or '101' according to the next event.

On a de Bruijn graph, we construct edge probabilities based on historical information. To achieve this, each node has two counters, c_0 for the number of *0*-events and c_1 for the number of *1*-events. These counters are incremented based on the corresponding event. Figure 13 shows a snapshot of the counters for state '001'. Based on the counter values, edge probabilities are computed, as shown in the figure (left-side).

By configuring max number of tickets (*MaxTicket*), it is possible to limit the window length we wish to monitor. The window length should be equivalent to *time frame length* $\times$ *MaxTicket*, where time frame length is an observation interval, in which a single event (zero or one) is generated based on the observation. If the total number of tickets is smaller than

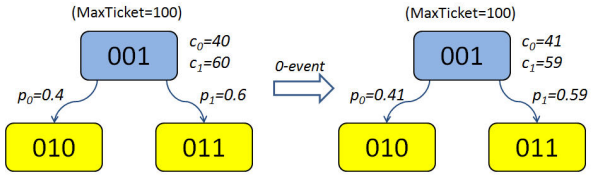


Fig. 13. Counters in a de Bruijn node: In our model, each de Bruijn node contains two counters for zero counter (c_0) and one counter (c_1). Any successive event changes the values of the counters, thus changing edge probabilities accordingly.

MaxTicket, the counters are simply incremented according to the event. After the total number of tickets reaches *MaxTicket*, however, one ticket in a counter is transferred to the other counter, instead of incrementing the counter. Thus, there will be no change in the total number of tickets (=*MaxTicket*) after this happens. In Figure 13, an *0*-event occurs, and we see that a ticket in c_1 is transferred to c_0 and the probabilities are recomputed accordingly (in the right-side figure). Note that the values of the counters are either a zero or a positive integer.

D. Gear-shift algorithm

Now, we present the FREP gear-shift algorithm. For down-shift, FREP relies on the above prediction model, while it uses a reactive model for up-shift. We first discuss how FREP determines down-shift, and then discuss up-shift.

FREP maintains a de Bruijn graph for each partition. To construct edge probabilities, we assume a *0*-event happened if the measured information meets the service constraint (e.g., 99% of requests are less than 500 ms) in the time frame (or *epoch*). On the other hand, if the percentage of violations is greater than the given percentile, we assume *1*-event happened. That is,

$$event = \begin{cases} 0 & \text{if } R(p) \leq \tau; \\ 1 & \text{otherwise.} \end{cases}$$

To determine down-shift, we calculate the probability of consecutive k zeros (i.e., the probability that the service constraint will be met for the following k time frames) at every epoch. If the computed probability is greater than a certain threshold (or *confidence*), we consider that the down-shift condition is satisfied, and the node with the highest index among active non-CS nodes will be sent to the standby mode. Naturally, no down-shift test is performed at the lowest gear level.

For clarity, we formally describe this procedure as follows. Let S_i be state i in the graph configured with k bits. Hence, there are max 2^k states, and we assume that i is the state number; for example S_0 indicates state '000', while S_7 is for state '111'. Let $P_{i,0}$ be the probability of *0*-edge at S_i, and similarly $P_{i,1}$ be the probability of *1*-edge at S_i. If we suppose the current state is S_a, the probability for consecutive k-zeros means the probability of transition from S_a to S_0, right after k time frames. Then, the probability $\mathcal{P}$ is defined:

$$\mathcal{P} = \prod_{i=0}^{k-1} P_{a \ll i,0} \tag{11}$$

Here, '$\ll$' is the bitwise shift-left operator. With the resulted probability, we decide the gear level:

$$Gear = \begin{cases} Gear - 1 & \text{if } \mathcal{P} \geq confidence; \\ Gear & \text{otherwise.} \end{cases}$$

We obtain the k value from the break-even time (T_{BE}). For the time frame size T_W, we set $k = \lceil T_{BE}/T_W \rceil$. The intuition behind this is that there is no energy penalty if the following k consecutive time frames satisfy the service constraint. However, prediction can sometimes fail. This can also happen for reasons such as a sudden change in workload characteristic. In such a case, we give a *penalty*, and edge probabilities are recomputed. With a penalty, all *0*-counters in the graph are dropped by half, and the tickets are transferred to the corresponding *1*-counters (regardless of the current total number of tickets). This decreases *0*-probabilities, resulting in more conservative down-shift decisions thereafter. For choosing T_W and *confidence* values, we explore impact of those parameters in the evaluation section (see Section V-E).

Making up-shift decisions relies on *reactive* information. In our mechanism, FREP immediately up-shifts the gear whenever it sees l consecutive misses against the service constraint, so as to prevent undesired performance degradation. In this paper, we used $l = 2$ by default to prevent any impulsive up-shift decision due to a temporal degradation. However, l may have a certain correlation with time frame size T_W. Investigation of this would be interesting and planned for future work. We may consider proactive up-shifts based on probabilities as well, but we have not seen any advantages for FREP.

V. Evaluation

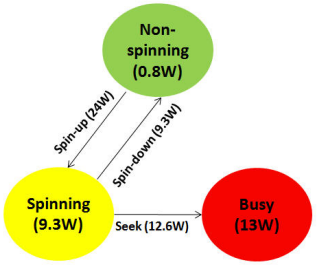
In this section we present an evaluation of FREP in terms of energy benefits and performance guarantees. We first focus on energy benefits (with relaxed constraints), and then discuss performance guarantees (with tight constraints). Before reporting our results, we describe our experimental setting and methodology in brief.

A. Experimental Setup

For evaluation, we augmented Disksim [25] which is widely used for studying storage systems. We considered Seagate Cheetah 15K.5 enterprise disks.[3] For this disk model, however, some power information, such as standby power and spin up/down power, is missing in the associated documents. For this reason, we alternatively chose power parameters from Seagate Barracuda specification.[4] Since the main purpose of our experiments here is to see applicability of FREP in terms of

[3]http://www.seagate.com/www/en-us/products/enterprise-hard-drives/cheetah-15k/

[4]http://www.seagate.com/support/disc/manuals/sata/100402371a.pdf



Description	Value
Spinning power	9.3 W
Busy power	13 W
Non-spinning power	0.8 W
Spin-up power	24 W
Spin-down power	9.3 W
Spin-up time	15 sec
Spin-down time	10 sec

Fig. 14. Disk power model (from Seagate Barracuda 7200)

both performance and power, we believe that comparing FREP with existing techniques with identical power parameters is a fair comparison. The power model we used in this paper is shown in Figure 14.

We assumed a datacenter environment with 120 disks. Although our model has no dependency on any specific RAID organization, we used RAID-5 structure as a unit of energy management. That is, a RAID-5 array is a node in our terminology. Each array has 4 data disks and 1 parity disk. Thus, there are 24 RAID arrays in the system (i.e., 24 nodes). We divided the system into 4 partitions, each of which consists of 2 CS nodes and 4 non-CS nodes. However, we also conducted experiments with different partition configurations in order to examine configuration effects.

We used multiple trace data, including real and synthetic traces: 2 Cello99 traces from HP Storage Research Lab [14]: a 1-day trace on May 1st (labeled "cello-1") and a 3-day trace between November 15th–17th (labeled "cello-2"); 2 financial traces provided by University of Massachusetts [15], each of which is labeled "umass-1" and "umass-2", respectively. The average inter-arrival time for Cello99 traces is 29.6 ms and 20.9 ms for cello-1 and cello-2, while it is 8.2 ms and 11.1 ms for umass-1 and umass-2, respectively.

To map the Cello99 traces to our configuration with 120 disks, we assumed that each disk in the traces is mapped to a single RAID array. Thus, 24 disks in the traces are mapped to 120 disks in 24 RAID arrays. The umass traces have no disk information. To use these traces in our experiments, we assumed that each application runs with a dedicated RAID array. Similarly to the Cello99 traces, umass requests are then mapped to RAID addresses.

We additionally created 9 synthetic workloads with different characteristics, as summarized in Table II. In the table, "exp(μ)" stands for exponential distribution with mean μ. For example, exp(6) for arrival distribution represents an exponential distribution with μ=6ms. The exp(6) and exp(20) arrival rates were characterized in Hibernator [7], and we added one more arrival distribution with exp(50) to consider an environment with a relative light load . It is observed in the literature that Internet data access patterns are related to a Zipf distribution with skewness α=1.0 [26]. In addition, the authors in [27] observed more heavily skewed accesses with α=1.8. We modeled synthetic traces based on those observations, in addition to uniform access distribution.

TABLE II
SYNTHETIC TRACES

Trace	# req.	Arrival dist.	Disk access dist.	# block dist.
S11	1 M	exp(6)	uniform	exp(20)
S12	1 M	exp(20)	uniform	exp(20)
S13	1 M	exp(50)	uniform	exp(20)
S21	1 M	exp(6)	Zipf(α=1.0)	exp(20)
S22	1 M	exp(20)	Zipf(α=1.0)	exp(20)
S23	1 M	exp(50)	Zipf(α=1.0)	exp(20)
S31	1 M	exp(6)	Zipf(α=1.8)	exp(20)
S32	1 M	exp(20)	Zipf(α=1.8)	exp(20)
S33	1 M	exp(50)	Zipf(α=1.8)	exp(20)

We evaluated 4 different systems: NPS is a base system for comparison without energy management; FTH is a system employing a fixed idleness threshold based on the 2-competitive algorithm; PARAID(k, l) is a PARAID configuration with a total of l disks with gear shifting down to k (thus, $l-k$ disks can go standby); and FREP(n, m) is an FREP configuration with a total of n nodes and m CS nodes in a partition. We set up two PARAID systems (PARAID(5,3) and PARAID(5,2)) and multiple FREP systems with different configurations, but mainly discuss the FREP(6,2) configuration. Thus, there are 4 partitions for 24 nodes for FREP(6,2) setting. By definition, PARAID systems can spin down disks with up to 40% (PARAID(5,3)) and 60% (PARAID(5,2)) of the total disks spun down, while FREP(6,2) can spin down 67% disks at max. We observed that PARAID(5,1), that allows spinning down of up to 80% disks, is severely degraded in terms of response time performance. For example, PARAID(5,1) increased average response time by a factor of 5 as compared to NPS with cello-1 trace. We thus excluded this configuration from our experiments.

As discussed in Section IV, FREP maintains de Bruijn graphs for gear-switch decisions. For the graphs, we used 5-second time frame (i.e., $T_{TF} = 5s$). Since we configure the number of bits based on the break-even time (i.e., number of bits = $\lceil T_{BE}/T_{TF} \rceil$), the graph is configured with 11 bits, since $T_{BE} = 54s$ based on the power model (Figure 14). In addition, we conservatively chose *confidence=0.9* for down-shift, and set consecutive miss counter *l=2* for up-shift to consider temporal performance degradation. We discuss the effects of time frame size and confidence in Section V-E.

We first present experimental results under relaxed service constraints to see the upper bound of energy benefits. After then, we will show how well FREP performs energy management satisfying the given service constraint.

B. Under relaxed service constraints

As discussed, FREP makes gear-shift decisions under consideration of service constraints. Here, we consider *relaxed constraints*. To give a relaxed constraint, we reverse-engineered NPS logs, then we set service constraints with values greater than twice of the NPS percentile numbers. For

example, 99% response time in the NPS results with cello-1 is 1,485 ms, and we used a number greater than 2×1485 ms for the 99% constraint for relaxation. Thus, FREP focuses more on energy saving in this case.

Figure 15 compares those two metrics with the real traces. Overall, FTH is fairly sensitive to workloads with respect to energy saving, and it shows very poor response times due to spin-up delay. PARAID and FREP consistently saved energy for different workloads. However, PARAID shows higher degree of variation in response time, while FREP shows fairly stable results by adaptively shifting gears to the given workload. Interestingly, we can see that FREP yields better response time than NPS with the cello-2 trace. This can be explained by an effect of request redirection. Bursty requests to a single node can be smoothed by redirecting them to multiple nodes when FREP is operating in energy saving mode.

Figure 16 shows the experimental results with a set of synthetic workloads. With these traces, FTH could make 0%–20% energy saving, but mean response time is very poor. Although not shown in the figure, mean response time in the worst case was 389 ms for S33, which is 100 times greater than the NPS's. The replication-based solutions (i.e., PARAID and FREP) are consistent, yielding significant energy saving with little performance loss.

C. Under tight service constraints

We next examine FREP with tight service constraints to see performance guarantees. We assume the following three types of constraints, based on our constraint model $R(p) \leq \tau$.

- *C1*: $(p = 0.9) \wedge (\tau = 90\%$ NPS response time);
- *C2*: $(p = 0.95) \wedge (\tau = 95\%$ NPS response time);
- *C3*: $(p = 0.99) \wedge (\tau = 99\%$ NPS response time).

For comparison, we call relaxed constraint *C0*.

Figure 17 shows the experimental results under tight constraints. In the figure, $P(n, m) \equiv PARAID(n, m)$. We can see that significant percentages of violations occurred for PARAID in order to obtain energy benefits for all three constraints. PARAID(5,2) shows heavy violations between 37%–57%. Even in the case of PARAID(5,3), the violations were over 20% for those real-world traces. FTH also shows some degree of violations greater than the constraints but smaller than PARAID. In contrast, FREP largely satisfies the given constraints. With tight constraints, FREP operates energy management more conservatively. Nonetheless, we can see that FREP still achieves non-trivial energy savings. FREP yields 15%–60% energy saving for the cello traces and 3%–15% for the umass traces.

Figure 18 shows the results with the synthetic traces for *C3* constraints. FREP successfully maintains violation rates to less than the 1% constraint. However, it yields energy saving only with skewed traces (i.e., S31 and S32).

Summarizing, although energy benefits achieved by FREP in this case may be reduced, it still provides strong performance guarantees. We observed only a single case that slightly exceeds the given constraint (5.4% for 5% requirement for cello-1 $C2$) out of 21 experimental cases.

TABLE III
ENERGY SAVING RATES

FREP setting	(4,1)	(4,2)	(6,1)	(6,2)	(6,3)
Theoretical max	75%	50%	83%	67%	50%
cello-1	68.4%	45.6%	75.9%	60.8%	45.6%
umass-1	68.2%	45.5%	75.3%	60.6%	45.5%

D. Impact of partition configuration

We next discuss the impact of FREP configuration. In this experiment, we configured 4 additional FREP settings with different numbers of nodes and CS nodes for each partition: FREP(4,1), FREP(4,2), FREP(6,1), and FREP(6,2). There may be various other possible configurations, but this result may be indicative for other configurations.

Figure 19 shows the experimental results with cello-1 under different partition settings. We can see that FREP saves energy from 40% to 76% compared to NPS. In terms of performance penalty, which is the ratio of the mean response time to the NPS's, it is only 14% at the worst case. Interestingly, 2 out of 5 cases showed negative penalty, which implies that FREP showed better response time than NPS. It is the impact of request redirection, as discussed in Section V-B.

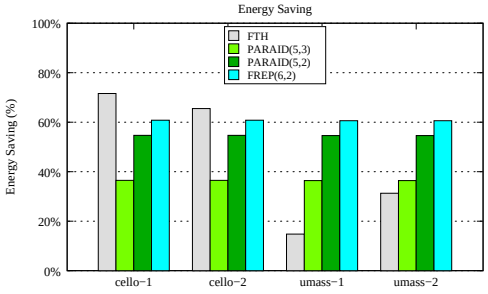
Table III summarizes energy saving rates to NPS in various FREP configurations. As shown in the table, FREP exploits replications, saving energy over 90% of the theoretical limits. Theoretical max is simply computed directly from the configuration setting. For example, theoretical max for FREP(4,1) is 75%, since 3 out of 4 nodes can be in the standby mode at max. The mean response times were less than 86.2 ms for cello-1 and less than 3.5 ms for umass-1, while NPS showed 75.7 ms and 0.9 ms, respectively.

E. Impact of time frame size and confidence

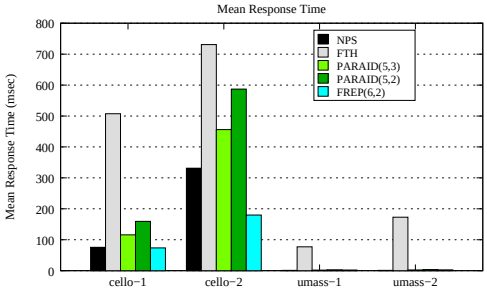
We next examined the impact of time frame size and confidence. The time frame size (T_W) determines the number of bits in the de Bruijn graph (with T_{BE}), while confidence (c) enables probabilistic decisions for down-shift on the graph. Greater T_W requires less space complexity for graph representation with smaller number of bits. As mentioned, we used $T_W = 5s$, and thus, the number of bits were 11. Additionally, we set up a graph with $T_W = 30s$, thus with 3 bits, for the graph. With those time frame sizes, we conducted experiments along different confidence values from $c = 0.1$ to $c = 0.9$ incrementing by 0.1 (we used $c = 0.9$ by default) under the constraint 1 (i.e., $C1$).

Figure 20 shows the results. The x-axis is confidence. The left y-axis shows energy saving to NPS, and the right y-axis shows the average response time in milliseconds. Overall, the results agree with our expectation. We can see that small time frame size yields greater energy saving, but worse response time. In contrast, the greater time frame size makes more conservative moves, thus yielding less energy saving but better response time.

Confidence largely affects both settings. With respect to response time, it becomes stable as $c \geq 0.8$ for $T_W = 5$.

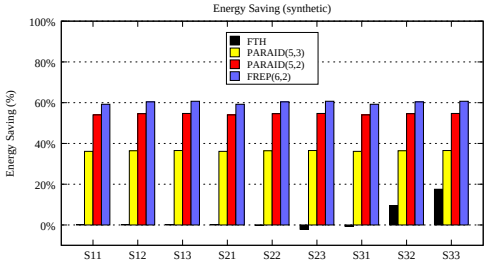


(a) Energy saving

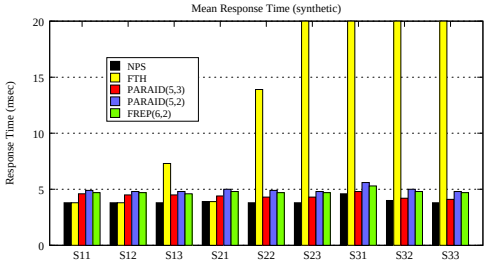


(b) Average response time

Fig. 15. Energy saving and average response time (real traces)

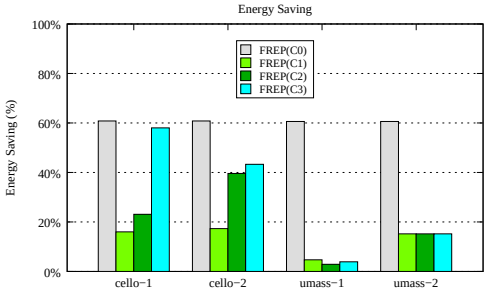


(a) Energy saving

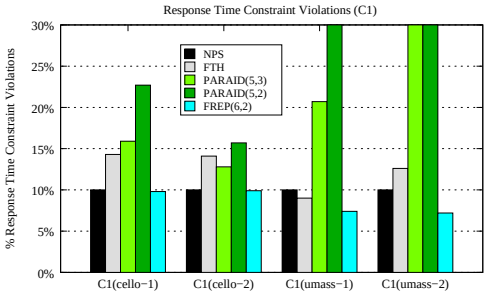


(b) Average response time

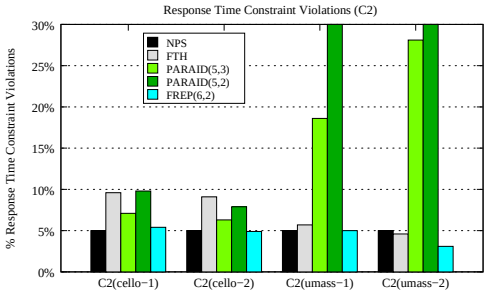
Fig. 16. Energy saving and average response time (Synthetic traces)



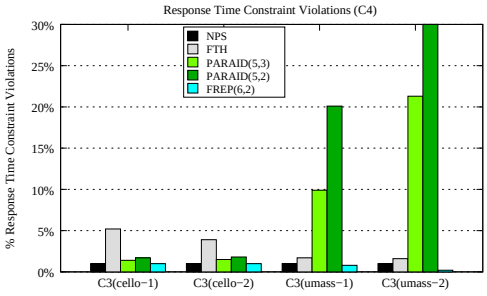
(a) Energy saving



(b) Percentage of response time constraint violation (C1)

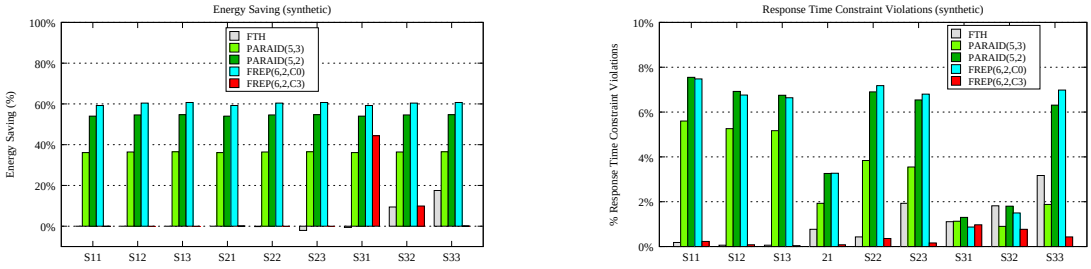


(c) Percentage of response time constraint violation (C2)



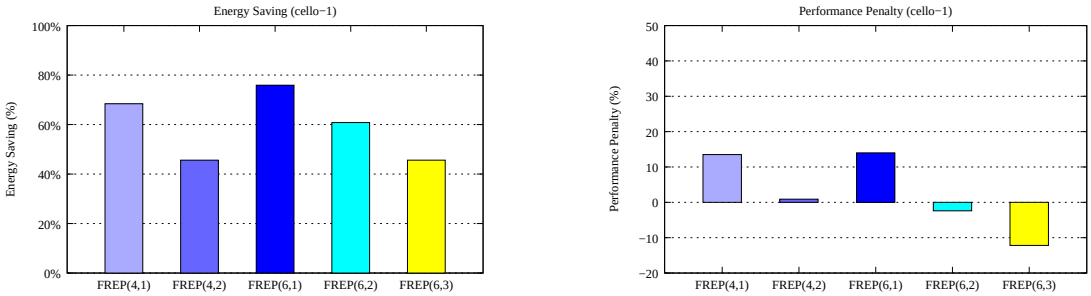
(d) Percentage of response time constraint violation (C3)

Fig. 17. Energy saving and performance guarantee under specified SLAs



(a) Energy saving (b) Percentage of response time constraint violation

Fig. 18. Energy saving and performance guarantees (Synthetic traces)



(a) Energy saving (b) Performance penalty

Fig. 19. Impact of FREP configurations with different number of CS and non-CS nodes (cello-1)

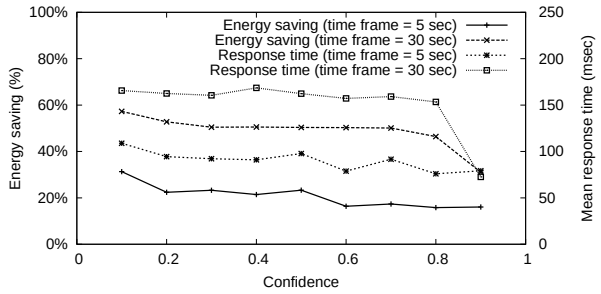


Fig. 20. Impact of time frame size and confidence (cello-1): Running with smaller time frame size results in more conservative energy management, yielding lesser energy saving but better response time. Confidence has smaller impact to smaller time frame size. Note that the y-axis is energy saving and the y2-axis is mean response time.

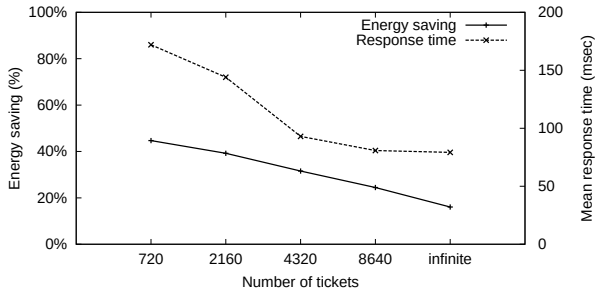


Fig. 21. Impact of number of tickets (cello-1): Running with smaller number of tickets results in more aggressive energy management. Note that the y-axis is energy saving and the y2-axis is mean response time.

In the case of $T_W = 30$, $c = 0.9$ improves response time dramatically. With less than that, there are no significant differences. In another experiment with different constraints with cello-1, we also observed that $c = 0.9$ under $T_W = 30$ yielded fairly good results. However, our intuition is that using a greater number of bits could guarantee performance more reliably.

F. Impact of the number of tickets

It is possible to configure *MaxTicket* (i.e., the maximum number of tickets) to set the window size. Intuitively, any large window size may be beneficial to construct more accurate probabilities, but may not be helpful for frequently varying workloads over time. With a small window size, in contrast, it may be possible to have hasty decisions with a small number of observations, although it can better explain recent workload characteristics.

Figure 21 shows the impact of *MaxTicket* with the cello-1 trace. In the x-axis, we varied the maximum number of tickets. As in Figure 20, the left y-axis is energy saving, while the right y-axis is the average response time. Overall, using a small window achieves better energy saving but greater response times, and vice versa. This indicates that using a small window works more aggressively, whereas using a large one works more conservatively for energy saving. Thus, *MaxTicket* can be tailored based on system goals and expected degree of workload variations.

G. Applicability of the gear-shift mechanism

We observed a high degree of violations for PARAID under tight constraints. This is due to the fact that PARAID refers only to disk utilization for its gear switching, and it

is questionable whether disk utilization is directly relevant to current load. For example, it can be underestimated due to a high rate of cache hits, and gear-shift decisions can be inadequate in this case because there may be a large number of outstanding requests in the input queue.

Our gear-shift mechanism is based on prediction by learning from the past history for better adaptability to workloads, and we have seen that it provides performance guarantees for given constraints. We applied this mechanism to PARAID and called it "PARAID*". The gear-switching operations were exactly the same as the FREP's. Figure 22 shows the PARAID* results with the cello-1 trace. For this experiment, we used two PARAID configurations, (5,2) and (5,3), with the three constraints (*C1–C3*). We can see that PARAID* provides fairly good performance guarantees, as well as energy savings. This suggests that our gear-shift technique is applicable to other systems with a gradual energy management function for energy benefits with performance guarantees.

VI. Conclusion

Energy proportionality is one of key metrics for future datacenters for both energy conservation and performance guarantees. In this work, we presented a technique called FREP (Fractional Replication for Energy Proportionality), for energy management that enhances energy proportionality in large datacenters. FREP includes a replication strategy and basic functions to enable flexible energy management. Specifically, our method provides performance guarantees by adaptively controlling the power states of a group of disks based on observed and predicted workloads. Our extensive experimental results with a broad set of traces showed that our energy management technique can achieve energy saving of over 90% of theoretical limits with little performance loss. With tight service constraints, we showed that FREP satisfies service constraints in diverse settings.

We plan to continue to work on our energy management algorithms and incorporate them in existing and newly created file systems. For example, Hadoop [28], a popular distributed computing framework, provides replication based on reconfigurable replication factor (without skewness). Our future work will investigate how we can incorporate techniques that guarantee energy proportionality while satisfying selected replication factors.

Acknowledgments

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] "http://www.federalnewsradio.com/pdfs/epadatacenterreporttocongress-august2007.pdf."

[2] J. Hamilton, "Cooperative expendable micro-slice servers (cems): Low cost, low power servers for internet-scale services," in *CIDR*, 2009.

[3] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke, "Reducing disk power consumption in servers with drpm," *Computer*, vol. 36, no. 12, pp. 59–66, 2003.

[4] S. Irani, G. Singh, S. K. Shukla, and R. K. Gupta, "An overview of the competitive and adversarial approaches to designing dynamic power management strategies," *IEEE Trans. VLSI Syst.*, vol. 13, no. 12, pp. 1349–1361, 2005.

[5] C. Weddle, M. Oldham, J. Qian, A.-I. A. Wang, P. Reiher, and G. Kuenning, "Paraid: A gear-shifting power-aware raid," *Trans. Storage*, vol. 3, no. 3, p. 13, 2007.

[6] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke, "Drpm: dynamic speed control for power management in server class disks," in *ISCA '03: Proceedings of the 30th annual international symposium on Computer architecture*. New York, NY, USA: ACM, 2003, pp. 169–181.

[7] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes, "Hibernator: helping disk arrays sleep through the winter," in *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 2005, pp. 177–190.

[8] T. Xie, "Sea: A striping-based energy-aware strategy for data placement in raid-structured storage systems," *IEEE Trans. Comput.*, vol. 57, no. 6, pp. 748–761, 2008.

[9] "D. Borthakur. The Hadoop Distributed File System: Architecture and Design., http://hadoop.apache.org/ core/docs/current/hdfs design.pdf."

[10] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 29–43, 2003.

[11] W. Lang, J. M. Patel, and J. F. Naughton, "On energy management, load balancing and replication," *SIGMOD Rec.*, vol. 38, no. 4, pp. 35–42, 2009.

[12] D. Li and J. Wang, "eraid: A queueing model based energy saving policy," in *MASCOTS '06: Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 77–86.

[13] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *IEEE Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[14] "Tools and traces, http://www.hpl.hp.com/research/ssp/software/."

[15] "Umass Trace Repository: OLTP Application I/O, http://traces.cs.umass.edu/index.php/storage/storage."

[16] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," in *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 2001, pp. 103–116.

[17] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," pp. 231–248, 2002.

[18] M. Elnozahy, M. Kistler, and R. Rajamony, "Energy conservation policies for web servers," in *USITS'03: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*. Berkeley, CA, USA: USENIX Association, 2003, pp. 8–8.

[19] D. P. Helmbold, D. D. E. Long, and B. Sherrod, "A dynamic disk spin-down technique for mobile computing," in *MobiCom '96: Proceedings of the 2nd annual international conference on Mobile computing and networking*. New York, NY, USA: ACM, 1996, pp. 130–142.

[20] G. Dhiman and T. S. Rosing, "Dynamic power management using machine learning," in *ICCAD '06: Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*. New York, NY, USA: ACM, 2006, pp. 747–754.

[21] E.-Y. Chung, L. Benini, A. Bogliolo, Y.-H. Lu, and G. De Micheli, "Dynamic power management for nonstationary service requests," *IEEE Trans. Comput.*, vol. 51, no. 11, pp. 1345–1361, 2002.

[22] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *Trans. Storage*, vol. 4, no. 3, pp. 1–23, 2008.

[23] L. Ganesh, H. Weatherspoon, M. Balakrishnan, and K. Birman, "Optimizing power consumption in large scale storage systems," in *HotOS*, 2007.

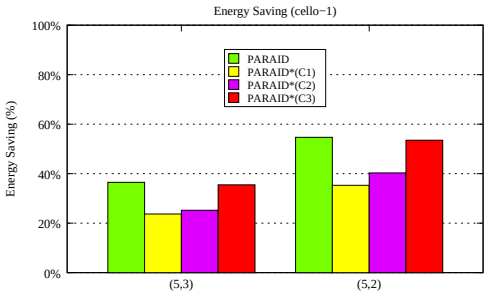
[24] A. Verma, R. Koller, L. Useche, and R. Rangaswami, "Srcmap: Energy proportional storage using dynamic consolidation," in *FAST*, 2010, pp. 267–280.

[25] "Disksim, http://www.pdl.cmu.edu/disksim/."

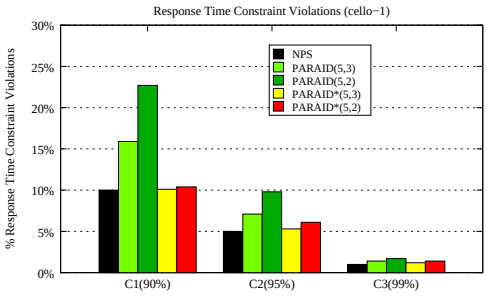
[26] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *Proceedings of INFOCOM (INFOCOM '99)*, 1999, pp. 126–134.

[27] V. N. Padmanabhan and L. Qiu, "The content and access dynamics of a busy web site: findings and implications," *SIGCOMM Comput. Commun. Rev.*, vol. 30, no. 4, pp. 111–123, 2000.

[28] "Hadoop:, http://hadoop.apache.org/."



(a) Energy saving



(b) Percentage of response time constraint violation

Fig. 22. PARAID with our gear-shift technique (cello-1)